



# PROGRAMACIÓN II – UNIDAD 2

## PYTHON 1

# Lenguaje python

## HOLA MUNDO Y COMENTARIOS

En un archivo llamado **holamundo.py** escribimos...

```
print("Hola mundo") #Comentario de línea
```

```
"""Comentario multi-  
línea"""
```

# Lenguaje Python

En Python los tipos básicos se dividen en:

- Números, como pueden ser 3 (entero), 15.57 (de coma flotante) o  $7 + 5j$  (complejos)
- Cadenas de texto, como "Hola Mundo"
- Valores booleanos: True (cierto) y False (falso).

```
# esto es una cadena  
c = "Hola Mundo"
```

```
# y esto es un entero  
e = 23
```

```
# podemos comprobarlo con la función type  
type(c)  
type(e)
```

Tipo de texto:

`str`

Tipos numéricos:

`int`, `float`, `complex`

Tipos de  
secuencia:

`list`, `tuple`, `range`

Tipo de mapeo:

`dict`

Tipos de  
conjuntos:

`set`, `frozenset`

Tipo booleano:

`bool`

Tipos binarios:

`bytes`, `bytearray`, `memoryview`

# Lenguaje Python

Puede haber ocasiones en las que desee especificar un tipo en una variable. Esto se puede hacer con yeso. Python es un lenguaje orientado a objetos y, como tal, usa clases para definir tipos de datos, incluidos sus tipos primitivos.

Por lo tanto, la conversión en Python se realiza mediante funciones constructoras:

- **int ()** : construye un número entero a partir de un literal entero, un literal flotante (redondeando hacia abajo al número entero anterior) o un literal de cadena (siempre que la cadena represente un número entero)
- **float ()** : construye un número flotante a partir de un literal entero, un literal flotante o un literal de cadena (siempre que la cadena represente un número flotante o entero)
- **str ()** : construye una cadena a partir de una amplia variedad de tipos de datos, incluidas cadenas, literales enteros y literales flotantes

```
x = int(1)      # x will be 1
y = int(2.8)    # y will be 2
z = int("3")    # z will be 3
```

```
x = float(1)    # x will be 1.0
y = float(2.8)  # y will be 2.8
z = float("3")  # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
x = str("s1")   # x will be 's1'
y = str(2)      # y will be '2'
z = str(3.0)    # z will be '3.0'
```

# Lenguaje python

## **Sintaxis para la conversión de tipos primitivos**

- `int (x)` Convierte x en un entero
- `long (x)` Convierte x en un entero largo
- `float (x)` Convierte x en un número de punto flotante
- `str (x)` Convierte x a una cadena. x puede ser del tipo float, entero o largo
- `hex (x)` Convierte x entero en una cadena hexadecimal
- `chr (x)` Convierte x entero a un caracter
- `ord (x)` Convierte el carácter x en un entero

# Lenguaje Python

## Sangría de Python

La sangría se refiere a los espacios al comienzo de una línea de código.

Mientras que en otros lenguajes de programación la sangría en el código es solo para legibilidad, la sangría en Python es muy importante. Python usa sangría para indicar un bloque de código.

```
if 7 > 3:  
    print("Siete supera a 3 !!!")
```

Esto daría error

```
if 5 > 2:  
print("Cinco supera a 2 !!!")
```

# Lenguaje Python

## Nombres de variables

Una variable puede tener un nombre corto (como xey) o un nombre más descriptivo (edad, nombre del coche, volumen\_total).

Reglas para variables de Python:

- El nombre de una variable **debe comenzar con una letra o el carácter de subrayado**
- El nombre de una variable **no puede comenzar con un número**
- El nombre de una variable **solo puede contener caracteres alfanuméricos y guiones bajos** (Az, 0-9 y \_)
- Los nombres de las variables **distinguen entre mayúsculas y minúsculas** (edad, Edad y EDAD son tres variables diferentes)

# Lenguaje python

## Cadenas

```
a = "uno"  
b = "dos"
```

```
c = a + b # c es "unodos"  
c = a * 3 # c es "unounouno"
```



# Lenguaje python

## Cadenas multilinea

Puede asignar una cadena de varias líneas a una variable utilizando tres comillas:

```
a = """Esto es una prueba sobre como  
se pueden generar  
Cadenas de multiples líneas"""  
  
print(a)
```

## Resultado

```
Esto es una prueba sobre como  
se pueden generar  
Cadenas de multiples líneas
```

# Lenguaje python

## Las cadenas son arrays

Como muchos otros lenguajes de programación populares, las cadenas en Python son matrices de bytes que representan caracteres Unicode.

Sin embargo, Python no tiene un tipo de datos de carácter, un solo carácter es simplemente una cadena con una longitud de 1.

Se pueden utilizar corchetes para acceder a elementos de la cadena.

```
a = "Hello, World!"  
print(a[1])
```

A terminal window with a black background and a white border. The character 'e' is displayed in white text, representing the output of the Python code snippet to its left.

e

# Lenguaje python

## Slicing

Puede devolver un rango de caracteres utilizando la sintaxis de sector.

Especifique el índice inicial y el índice final, separados por dos puntos, para devolver una parte de la cadena.

Obtenga los caracteres de la posición 2 a la posición 5 (no incluidos):

```
b = "Hello, World!"  
print(b[2:5])
```



```
llo
```

# Lenguaje python

## Indexación negativa

Use índices negativos para comenzar el segmento desde el final de la cadena.

Obtenga los caracteres de la posición 5 a la posición 1 (no incluido), comenzando el conteo desde el final de la cadena:

```
b = "Hello, World!"  
print(b[-5:-2])
```

```
orl
```

# Lenguaje python

## Longitud de la cadena

Para obtener la longitud de una cadena, use la función len().

```
a = "Hello, World!"  
print(len(a))
```

```
13
```

## Métodos de cadena

Python tiene un conjunto de métodos integrados que puede usar en cadenas.

El método strip() elimina cualquier espacio en blanco del principio o del final:

```
a = " Hello, World! "  
print(a.strip())
```

```
Hello, World!
```

# Lenguaje python

El método `lower()` devuelve la cadena en minúsculas.

```
a = "Hello, World!"  
print(a.lower())
```

```
hello, world!
```

El método `upper()` devuelve la cadena en mayúsculas.

```
a = "Hello, World!"  
print(a.upper())
```

```
HELLO, WORLD!
```

El método `replace()` reemplaza una cadena con otra cadena.

```
a = "Hello, World!"  
print(a.replace("H", "J"))
```

```
Jello, World!
```

El método `split()` divide la cadena en subcadenas si encuentra instancias del separador.

```
a = "Hello, World!"  
b = a.split(",")  
print(b)
```

```
['Hello', ' World!']
```

Para comprobar si una determinada frase o carácter está presente en una cadena, podemos utilizar las palabras clave **in** o **not in**.

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)
```

```
True
```

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" not in txt  
print(x)
```

```
False
```

# Lenguaje python

## Formato de cadena

Podemos combinar cadenas y números usando el método format().

El método format() toma los argumentos pasados, los formatea y los coloca en la cadena donde están los marcadores de posición {}:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

# Lenguaje python

## Formato de cadena

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

Puede usar números índice {0} para asegurarse de que los argumentos se coloquen en los marcadores en la posición correctos:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567
```



# Lenguaje python

## Caracteres de Escape:

Para insertar caracteres que son ilegales en una cadena, use un carácter de escape. Un carácter de escape es una barra invertida \ seguida del carácter que desea insertar. Un ejemplo de carácter ilegal es una comilla doble dentro de una cadena rodeada de comillas dobles:

El carácter de escape le permite usar comillas dobles cuando normalmente no estaría permitido:

```
txt = "We are the so-called \"Vikings\" from the north."  
print(txt)
```

```
We are the so-called "Vikings" from the north.
```

# Lenguaje python

## Caracteres de Escape

Code	Result
<code>\'</code>	Single Quote
<code>\\</code>	Backslash
<code>\n</code>	New Line
<code>\r</code>	Carriage Return
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\f</code>	Form Feed
<code>\ooo</code>	Octal value
<code>\xhh</code>	Hex value

# Lenguaje python

## Métodos para los string

`capitalize()`

Convierte la primera letra a mayúscula

```
txt = "hello, and welcome to my world."  
  
x = txt.capitalize()  
  
print (x)
```

```
Hello, and welcome to my world.
```

# Lenguaje python

## Métodos para los string

`casefold()`

Convierte el texto a minúsculas

```
txt = "Hello, And Welcome To My World!"  
  
x = txt.casefold()  
  
print(x)
```

```
hello, and welcome to my world!
```

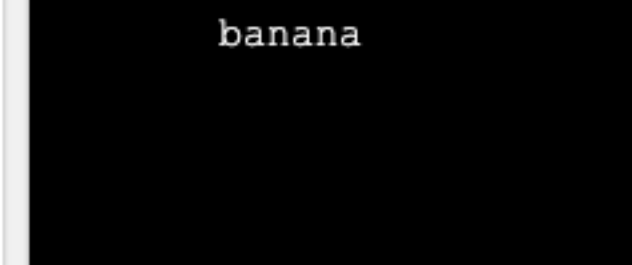
# Lenguaje python

## Métodos para los string

`center()`

Retirna un string centrado

```
txt = "banana"  
  
x = txt.center(20)  
  
print(x)|
```



banana

# Lenguaje python

## Métodos para los string

count()

Retorna cuantas veces aparece un carácter en una cadena

```
txt = "I love apples, apple are my favorite fruit"  
  
x = txt.count("apple")  
  
print(x)
```

2

# Lenguaje python

## Métodos para los string

`encode()`

Retorna una versión codificada del string

```
txt = "My name is Ståle"  
  
x = txt.encode()  
  
print(x)
```

```
b'My name is St\xc3\xe5le'
```

# Lenguaje python

## Métodos para los string

endswith()

Retorna verdadero si la cadena termina con el valor especificado

```
txt = "Hello, welcome to my world."  
  
x = txt.endswith(".")  
  
print(x)
```

True



# Lenguaje python

## Métodos para los string

[expandtabs\(\)](#)

Establece el valor del tab que se utilizará en el string

```
txt = "H\te\tl\tl\to"  
x = txt.expandtabs(4)  
print(x)
```



H e l l o

# Lenguaje python

## Métodos para los string

find()

Busca un valor en el string y retorna cuál es su posición

```
txt = "Hello, welcome to my world."  
  
x = txt.find("welcome")  
  
print(x)  
|
```

7

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

```
txt = "For only {price:.2f} dollars!"  
print(txt.format(price = 49))
```

```
For only 49.00 dollars!
```

```
#To demonstrate, we insert the number 8 to set the  
available space for the value to 8 characters.
```

```
#Use "<" to left-align the value:
```

```
txt = "We have {:<8} chickens."  
print(txt.format(49))
```

```
We have 49      chickens.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

```
#To demonstrate, we insert the number 8 to set the  
available space for the value to 8 characters.
```

```
#Use ">" to right-align the value:
```

```
txt = "We have {:>8} chickens."  
print(txt.format(49))
```

```
We have      49 chickens.
```

```
#To demonstrate, we insert the number 8 to set the  
available space for the value to 8 characters.
```

```
#Use "^" to center-align the value:
```

```
txt = "We have {:^8} chickens."  
print(txt.format(49))
```

```
We have      49      chickens.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

```
#To demonstrate, we insert the number 8 to specify the
available space for the value.
```

```
#Use "=" to place the plus/minus sign at the left most
position:
```

```
txt = "The temperature is {:s8} degrees celsius."

print(txt.format(-5))
```

```
The temperature is -      5 degrees.
```

```
#Use "+" to always indicate if the number is
positive or negative:
```

```
txt = "The temperature is between {:+} and {:+}
degrees celsius."

print(txt.format(-3, 7))
```

```
The temperature is between -3 and +7 degrees celsius.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

#Use "-" to always indicate if the number is negative (positive numbers are displayed without any sign):

```
txt = "The temperature is between {: -} and {: -} degrees celsius."
```

```
print(txt.format(-3, 7))
```

```
The temperature is between -3 and 7 degrees celsius.
```

#Use " " (a space) to insert a space before positive numbers and a minus sign before negative numbers:

```
txt = "The temperature is between {: } and {: } degrees celsius."
```

```
print(txt.format(-3, 7))
```

```
The temperature is between -3 and 7 degrees celsius.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

#Use "," to add a comma as a thousand separator:

```
txt = "The universe is {:,} years old."
```

```
print(txt.format(13800000000))
```

```
The universe is 13,800,000,000 years old.
```

#Use "\_" to add a underscore character as a thousand separator:

```
txt = "The universe is {:_} years old."
```

```
print(txt.format(13800000000))
```

```
The universe is 13_800_000_000 years old.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

#Use "b" to convert the number into binary format:

```
txt = "The binary version of {0} is {0:b}"
```

```
print(txt.format(5))
```

```
The binary version of 5 is 101
```

#Use "d" to convert a number, in this case a binary number, into decimal number format:

```
txt = "We have {:d} chickens."
```

```
print(txt.format(0b101))
```

```
We have 5 chickens.
```



# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

#Use "e" to convert a number into scientific number format (with a lower-case e):

```
txt = "We have {:e} chickens."  
print(txt.format(5))
```

```
We have 5.000000e+00.
```

#Use "E" to convert a number into scientific number format (with an upper-case E):

```
txt = "We have {:E} chickens."  
print(txt.format(5))
```

```
We have 5.000000E+00.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

#Use "f" to convert a number into a fixed point number, default with 6 decimals, but use a period followed by a number to specify the number of decimals:

```
txt = "The price is {:.2f} dollars."  
print(txt.format(45))
```

#without the ".2" inside the placeholder, this number will be displayed like this:

```
txt = "The price is {:f} dollars."  
print(txt.format(45))
```

```
The price is 45.000 dollars.  
The price is 45.000000 dollars.
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

`#Use "o" to convert the number into octal format:`

```
txt = "The octal version of {0} is {0:o}"
```

```
print(txt.format(10))
```

```
The octal version of 10 is 12
```

`#Use "x" to convert the number into Hex format:`

```
txt = "The Hexadecimal version of {0} is {0:x}"
```

```
print(txt.format(255))
```

```
The Hexadecimal version of 255 is ff
```

# Lenguaje python

## Métodos para los string

`format()`

Especifica el valor de formateo para un string

**#Use "X" to convert the number into upper-case Hex format:**

```
txt = "The Hexadecimal version of {0} is {0:X}"  
print(txt.format(255))
```

```
The Hexadecimal version of 255 is FF
```

**#Use "%" to convert the number into a percentage format:**

```
txt = "You scored {:.%}"  
print(txt.format(0.25))
```

**#Or, without any decimals:**

```
txt = "You scored {:.0%}"  
print(txt.format(0.25))
```

```
You scored 25.000000%  
You scored 25%
```

# Lenguaje python

## Métodos para los string

index()

Retorna la posición donde se encuentra el valor buscado dentro del string

Ejemplo

¿En qué parte del texto aparece la primera aparición de la letra "e" cuando solo busca entre la posición 5 y 15?:

```
txt = "Hello, welcome to my world."  
  
x = txt.index("e", 5, 15)  
  
print(x)
```

8

# Lenguaje python

## Métodos para los string

isalnum()

Retorna verdadero si todos los caracteres del string son alfanuméricos

```
txt = "Company12"  
  
x = txt.isalnum()  
  
print(x)
```

True

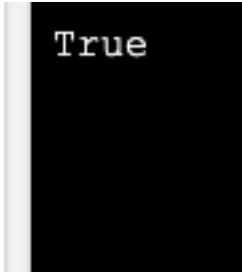
# Lenguaje python

## Métodos para los string

isalpha()

Retorna verdadero si todos los caracteres del string son alfabéticos

```
txt = "CompanyX"  
  
x = txt.isalpha()  
  
print(x)
```

A terminal window with a black background and white text. The word "True" is displayed in a monospaced font, indicating the output of the Python code execution.

True

# Lenguaje python

## Métodos para los string

isdecimal()

Retorna verdadero si todos los caracteres del string son decimales

```
a = "\u0030" #unicode for 0
b = "\u0047" #unicode for G

print(a.isdecimal())
print(b.isdecimal())
```

```
True
False
```



# Lenguaje python

## Métodos para los string

isdigit()

Retorna verdadero si todos los caracteres en el string son digitos

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for ²
c = "50800"
d = "50800a"
```

```
print(a.isdigit())
print(b.isdigit())
print(c.isdigit())
print(d.isdigit())
```

```
True
True
True
False
```

# Lenguaje python

## Métodos para los string

isidentifier()

Returns True if the string is an identifier

Una cadena se considera un identificador válido si solo contiene letras alfanuméricas (az) y (0-9), o guiones bajos (\_). Un identificador válido no puede comenzar con un número ni contener espacios.

```
a = "MyFolder"  
b = "Demo002"  
c = "2bring"  
d = "my demo"
```

```
print(a.isidentifier())  
print(b.isidentifier())  
print(c.isidentifier())  
print(d.isidentifier())
```

```
True  
True  
False  
False
```

# Lenguaje python

## Métodos para los string

islower()

Retorna verdadero si todos los caracteres del string son minúsculas

```
a = "Hello world!"  
b = "hello 123"  
c = "mynameisPeter"  
  
print(a.islower())  
print(b.islower())  
print(c.islower())
```

```
False  
True  
False
```

# Lenguaje python

## Métodos para los string

isnumeric()

Retorna verdadero si todos los valores del string son numéricos

```
a = "\u0030" #unicode for 0  
b = "\u00B2" #unicode for ²  
c = "10km2"
```

```
print(a.isnumeric())  
print(b.isnumeric())  
print(c.isnumeric())
```

```
True  
True  
False
```

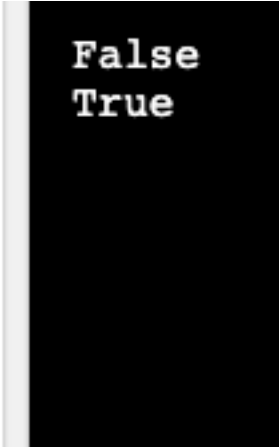
# Lenguaje python

## Métodos para los string

isprintable()

Retorna verdadero si todos los caracteres del string son imprimibles

```
txt1 = "Hello!\nAre you #1?"  
txt2 = "Hello! Are you #1?"  
  
x = txt1.isprintable()  
y = txt2.isprintable()  
  
print(x)  
print(y)
```



False  
True

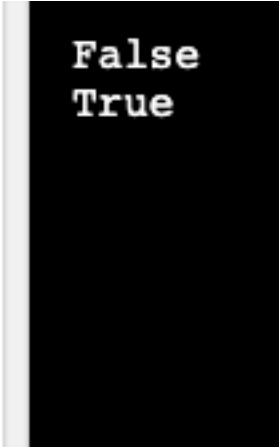
# Lenguaje python

## Métodos para los string

`isprintable()`

Retorna verdadero si todos los caracteres del string son imprimibles

```
txt1 = "Hello!\nAre you #1?"  
txt2 = "Hello! Are you #1?"  
  
x = txt1.isprintable()  
y = txt2.isprintable()  
  
print(x)  
print(y)
```

A terminal window with a black background and white text. The first line displays 'False' and the second line displays 'True', corresponding to the output of the Python code shown to the left.

False  
True

# Lenguaje python

## Métodos para los string

isspace()

Returns True if all characters in the string are whitespaces

```
txt1 = "  s  "  
txt2 = "      "  
  
x = txt1.isspace()  
y = txt2.isspace()  
  
print(x)  
print(y)
```

False  
True

# Lenguaje python

## Métodos para los string

istitle()

Retorna Verdadero si todas las palabras de un texto comienzan con una letra mayúscula Y el resto de la palabra son letras minúsculas; de lo contrario, Falso.

```
a = "HELLO, AND WELCOME TO MY WORLD"  
b = "Hello"  
c = "22 Names"  
d = "This Is %'!?"
```

```
print(a.istitle())  
print(b.istitle())  
print(c.istitle())  
print(d.istitle())
```

```
False  
True  
True  
True
```



# Lenguaje python

## Métodos para los string

isupper()

Retorna verdadero si todos los caracteres del string son mayúsculas

```
a = "Hello World!"  
b = "hello 123"  
c = "MY NAME IS PETER"  
  
print(a.isupper())  
print(b.isupper())  
print(c.isupper())
```

```
False  
False  
True
```

# Lenguaje python

## Métodos para los string

`join()`

Toma todos los elementos en un iterable y los une en una cadena.  
Se debe especificar una cadena como separador.

```
myTuple = ("John", "Peter", "Vicky")  
  
x = "#".join(myTuple)  
  
print(x)
```

```
John#Peter#Vicky
```

# Lenguaje python

## Métodos para los string

ljust()

alineará la cadena a la izquierda, utilizando un carácter especificado (el espacio es el predeterminado) como carácter de relleno.

```
txt1 = "banana"
txt2 = "banana"

x1 = txt1.ljust(20)
x2 = txt2.ljust(20, "-")

print(x1)
print(x2)
```



```
banana
banana-----
```


# Lenguaje python

## Métodos para los string

lower()

Convierte un string en minúsculas

```
txt = "Hello my FRIENDS"  
  
x = txt.lower()  
  
print(x)
```



```
hello my friends
```

# Lenguaje python

## Métodos para los string

`rstrip()`

Elimina los caracteres iniciales (el espacio es el carácter inicial predeterminado para eliminar)

```
txt1 = "        banana        "  
txt2 = ",,,,,ssaaww.....manzana"  
  
x1 = txt1.rstrip()  
x2 = txt2.rstrip(",.asw")  
  
print("of all fruits", x1, "is my favorite")  
print(x2)
```

```
of all fruits banana        is my favorite  
manzana
```

# Lenguaje python

## Métodos para los string

[maketrans\(\)](#)

método devuelve una tabla de mapeo que se puede usar con el método para reemplazar los caracteres especificados. [translate\(\)](#)

```
txt = "Hello Sam!";  
mytable = txt.maketrans("S", "P");  
print(txt.translate(mytable));
```



Hello Pam!

El tercer parámetro de la tabla de asignación describe los caracteres que desea eliminar de la cadena:

```
txt = "Good night Sam!";  
  
x = "mSa";  
y = "eJo";  
z = "odnght";  
  
mytable = txt.maketrans(x, y, z);  
print(txt.translate(mytable));
```



G i Joe!

# Lenguaje python

## Métodos para los string

`partition()`

Busca la palabra "bananas" y devuelve una tupla con tres elementos:

1 - todo antes de "bananas"

2 - el "bananas"

3 - todo después del "bananas"

```
txt = "I could eat bananas all day"  
x = txt.partition("bananas")  
print(x)
```

```
('I could eat ', 'bananas', ' all day')
```

```
txt = "I could eat bananas all day"  
x = txt.partition("apples")  
print(x)
```

```
('I could eat bananas all day', '', '')
```

# Lenguaje python

## Métodos para los string

[replace\(\)](#)

Reemplaza una frase especificada con otra frase especificada.

```
txt = "I like bananas"

x = txt.replace("bananas", "apples")

print(x)
```

```
I like apples
```

El tercer parámetro indica la cantidad de veces que se producirán los reemplazos

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three", 2)

print(x)
```

```
three three was a race horse, two two was one too."
```



# Lenguaje python

## Métodos para los string

### rfind()

Busca la última aparición del valor especificado.  
Retorna -1 si no se encuentra el valor.  
Es similar al método rindex() método.

```
txt = "Mi casa, su casa."  
x = txt.rfind("casa")  
print(x)
```

12

```
txt = "Hello, welcome to my world."  
x = txt.rfind("e", 5, 10)  
print(x)
```

8

### Valores paramétricos

Parameter	Description
<i>value</i>	Required. The value to search for
<i>start</i>	Optional. Where to start the search. Default is 0
<i>end</i>	Optional. Where to end the search. Default is to the end of the string

# Lenguaje python

## Métodos para los string

### `rindex()`

Busca la última aparición del valor especificado.  
Retorna una excepción si no se encuentra el valor.  
Es similar al método `rfind()` método.

```
txt = "Mi casa, su casa."  
  
x = txt.rindex("casa")  
  
print(x)
```

12

```
txt = "Hello, welcome to my world."  
  
x = txt.rindex("e", 5, 10)  
  
print(x)
```

8

# Lenguaje python

## Métodos para los string

`rjust()`

Retorna la versión justificada a la derecha del string

```
txt = "banana"

x = txt.rjust(20)

print(x, "is my favorite fruit.")
```

```
banana is my favorite fruit.
```

```
txt = "banana"

x = txt.rjust(20, "-")

print(x)
```

```
-----banana
```

# Lenguaje python

## Métodos para los string

### rpartition()

Busca la última aparición de la palabra "bananas" y devuelva una tupla con tres elementos:

- 1 - todo antes de "bananas"
- 2 - el "bananas"
- 3 - todo después del "bananas"

```
txt = "I could eat bananas all day,  
bananas are my favorite fruit"
```

```
x = txt.rpartition("bananas")
```

```
print(x)
```

```
('I could eat bananas all day, ', 'bananas', ' are my favorite fruit')
```

# Lenguaje python

## Métodos para los string

[rsplit\(\)](#)

Divida una cadena en una lista donde cada palabra es un elemento de lista

```
txt = "welcome to the jungle"  
  
x = txt.split()  
  
print(x)
```

```
['welcome', 'to', 'the', 'jungle']
```

```
txt = "hello, my name is Peter, I am 26  
years old"  
  
x = txt.split(", ")  
  
print(x)
```

```
['hello', 'my name is Peter', 'I am 26 years old']
```

# Lenguaje python

## Métodos para los string

`splitlines()`

Divide una cadena en una lista. La división se realiza en los saltos de línea.

```
txt = "Thank you for the music\nWelcome to the jungle"
x = txt.splitlines()
print(x)
```

```
['Thank you for the music', 'Welcome to the jungle']
```

El parámetro True / False se utiliza para determinar si se deja en \n como parte del texto

```
txt = "Thank you for the
music\nWelcome to the jungle"
x = txt.splitlines(True)
print(x)
```

```
['Thank you for the music\n', 'Welcome to the jungle']
```

# Lenguaje python

## Métodos para los string

startswith()

Retorna Verdadero si el string comienza con el valor especificado

```
txt = "Hello, welcome to my world."  
x = txt.startswith("Hello")  
print(x)
```

True

```
txt = "Hello, welcome to my world."  
x = txt.startswith("wel", 7, 20)  
print(x)
```

True

## Valores paramétricos

Parameter	Description
<i>value</i>	Required. The value to check if the string starts with
<i>start</i>	Optional. An Integer specifying at which position to start the search
<i>end</i>	Optional. An Integer specifying at which position to end the search

# Lenguaje python

## Métodos para los string

strip()

Elimina los espacios al principio y al final de la cadena

```
txt = "    banana    "  
x = txt.strip()  
print("of all fruits", x, "is my favorite")
```

```
of all fruits banana is my favorite
```

```
txt = ",,,,rrttgg....banana....rrr"  
x = txt.strip(",.grt")  
print(x)
```

```
banana
```



# Lenguaje python

## Métodos para los string

`swapcase()`

Convierte las letras minúsculas en mayúsculas y las mayúsculas en minúsculas

```
txt = "Hello My Name Is PETER"  
  
x = txt.swapcase()  
  
print(x)
```



```
hELLO mY nAME iS peter
```

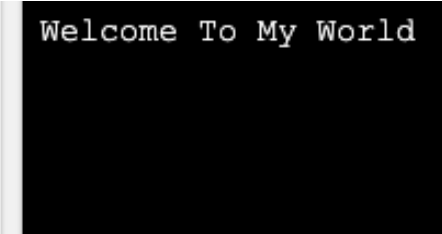
# Lenguaje python

## Métodos para los string

`title()`

Escribe la primera letra de cada palabra en mayúscula

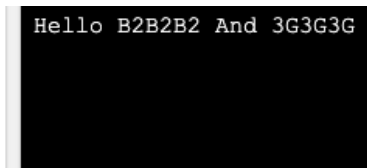
```
txt = "Welcome to my world"  
  
x = txt.title()  
  
print(x)
```



```
Welcome To My World
```

Tenga en cuenta que la primera letra después de una letra no alfabética se convierte en una letra mayúscula

```
txt = "hello b2b2b2 and 3g3g3g"  
  
x = txt.title()  
  
print(x)
```



```
Hello B2B2B2 And 3G3G3G
```

# Lenguaje python

## Métodos para los string

`translate()`

Retorna un string traducido

```
#use a dictionary with ascii codes to replace  
83 (S) with 80 (P):  
mydict = {83: 80};  
  
txt = "Hello Sam!";  
  
print(txt.translate(mydict));
```

A terminal window with a black background and white text. The text "Hello Pam!" is displayed on the first line. The terminal has a vertical scrollbar on the left side.

Hello Pam!

# Lenguaje python

## Métodos para los string

upper()

Convierte un strien a mayúscula

```
txt = "Hello my friends"  
  
x = txt.upper()  
  
print(x)
```



```
HELLO MY FRIENDS
```

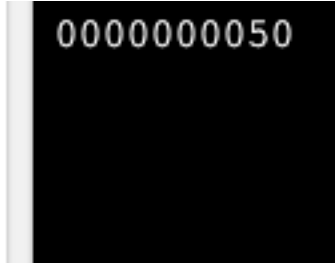
# Lenguaje python

## Métodos para los string

zfill()

Llena la cadena con ceros hasta el valor especificados

```
txt = "50"  
  
x = txt.zfill(10)  
  
print(x)
```



0000000050

# Lenguaje python



## Operadores de Python

Los operadores se utilizan para realizar operaciones sobre variables y valores. Python divide los operadores en los siguientes grupos:

- Operadores aritméticos
- Operadores de Asignación
- Operadores de comparación
- Operadores lógicos
- Operadores de identidad
- Operadores de membresía
- Operadores bit a bit

# Lenguaje python

## Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma	<code>r = 3 + 2</code> # r es 5
-	Resta	<code>r = 4 - 7</code> # r es -3
-	Negación	<code>r = -7</code> # r es -7
*	Multipliación	<code>r = 2 * 6</code> # r es 12
**	Exponente	<code>r = 2 ** 6</code> # r es 64
/	División	<code>r = 3.5 / 2</code> # r es 1.75
//	División entera	<code>r = 3.5 // 2</code> # r es 1.0
%	Módulo	<code>r = 7 % 2</code> # r es 1

# Lenguaje python

## Operadores De Asignación

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3



# Lenguaje python

## Operadores Lógicos que retornan valores bool

Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos, los llamados operadores lógicos o condicionales:

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	<code>r = True and False # r es False</code>
or	¿se cumple a o b?	<code>r = True or False # r es True</code>
not	No a	<code>r = not True # r es False</code>

# Lenguaje python

## Operadores de Comparación que retornan valores bool

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
<code>==</code>	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
<code>!=</code>	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<code>&lt;</code>	¿es a menor que b?	<code>r = 5 &lt; 3 # r es False</code>
<code>&gt;</code>	¿es a mayor que b?	<code>r = 5 &gt; 3 # r es True</code>

# Lenguaje python

## Operadores de identidad

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

# Lenguaje python

**Operadores de membresía:** Los operadores de pertenencia se utilizan para probar si se presenta una secuencia en un objeto

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
x = ["manzana", "banana"]
```

```
print("banana" in x)
```

```
# returns True because a sequence with  
the value "banana" is in the list
```

True

```
x = ["manzana", "banana"]
```

```
print("pera" not in x)
```

```
# returns True because a  
sequence with the value  
"pineapple" is not in the list
```

True

# Lenguaje python

## Operadores a nivel de bit

Operador	Descripción	Ejemplo
&	and	<code>r = 3 &amp; 2 # r es 2</code>
	or	<code>r = 3   2 # r es 3</code>
^	xor	<code>r = 3 ^ 2 # r es 1</code>
~	not	<code>r = ~3 # r es -4</code>
<<	Desplazamiento izq.	<code>r = 3 &lt;&lt; 1 # r es 6</code>
>>	Desplazamiento der.	<code>r = 3 &gt;&gt; 1 # r es 1</code>

# Lenguaje python

## Ejemplos de operadores bit a bit

```
1 x = bin(10)
2 y = bin(15)
3 a = int(x,base=2)
4 b = int(y,base=2)
5 print("10 Decimal es: " + x + " en binario")
6 print("15 Decimal es: " + y + " en binario")
7 c = bin(a & b)
8 d = bin(a | b)
9 e = bin(a ^ b)
10 f = bin(a << 1)
11 print("Resultado de aplicar el operador bit a bit & (and): " + c)
12 print("Resultado de aplicar el operador bit a bit | (or): " + d)
13 print("Resultado de aplicar el operador bit a bit ^ (xor): " + e)
14 print("Resultado de aplicar el operador bit a bit << (di): " + f)
```

```
10 Decimal es: 0b1010 en binario
15 Decimal es: 0b1111 en binario
Resultado de aplicar el operador bit a bit & (and): 0b1010
Resultado de aplicar el operador bit a bit | (or): 0b1111
Resultado de aplicar el operador bit a bit ^ (xor): 0b101
Resultado de aplicar el operador bit a bit << (di): 0b10100
> 
```

# Lenguaje python

## Formateo de salida

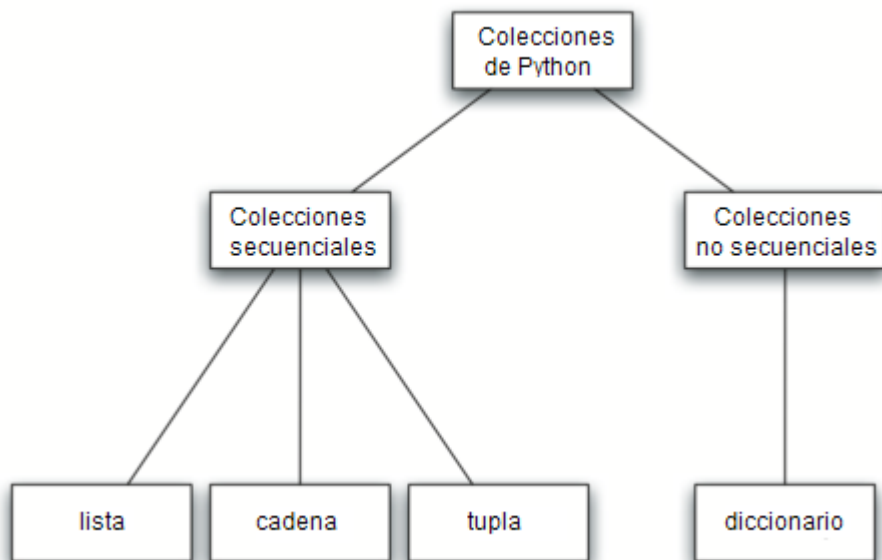
```
a = 8
b= "Hello"
c= True

#Formateo de salida...
print("Concatenando " + str(a) + " " + b + " " + str(c)) #Obligatorio hacer cast a string
print("Concatenando",a,b,c)
print("Concatenando %s %s %s" % (a,b,c))

my_string = "Concatenando {} {} {}"
print(my_string.format(a,b,c))

print(f"Concatenando {a} {b} {c} ") #Sugerida...
```

## COLECCIONES





# Lenguaje python

## Listas

La lista es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores.

Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, ... y también listas.

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista:

```
l = [22, True, "una lista", [1, 2]]
```

# Lenguaje python - Listas

Podemos acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes. Ten en cuenta sin embargo que el índice del primer elemento de la lista es 0, y no 1:

```
l = [11, False]
mi_var = l[0] # mi_var vale 11
```

Si queremos acceder a un elemento de una lista incluida dentro de otra lista tendremos que utilizar dos veces este operador, primero para indicar a qué posición de la lista exterior queremos acceder, y el segundo para seleccionar el elemento de la lista interior:

```
l = ["una lista", [1, 2]]
mi_var = l[1][0] # mi_var vale 1
```

# Lenguaje python - Listas

También podemos utilizar este operador para modificar un elemento de la lista si lo colocamos en la parte izquierda de una asignación:

```
l = [22, True]
l[0] = 99 # Con esto l valdrá [99, True]
```

Una curiosidad sobre el operador `[]` de Python es que podemos utilizar también números negativos. Si se utiliza un número negativo como índice, esto se traduce en que el índice empieza a contar desde el final, hacia la izquierda; es decir, con `[-1]` accederíamos al último elemento de la lista, con `[-2]` al penúltimo, con `[-3]`, al antepenúltimo, y así sucesivamente.

# Lenguaje python - Listas

Otra cosa inusual es lo que en Python se conoce como *slicing* o particionado, y que consiste en ampliar este mecanismo para permitir seleccionar porciones de la lista. Si en lugar de un número escribimos dos números inicio y fin separados por dos puntos (inicio:fin) Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, sin incluir este último. Si escribimos tres números (inicio:fin:salto) en lugar de dos, el tercero se utiliza para determinar cada cuantas posiciones añadir un elemento a la lista.

```
l = [99, True, "una lista", [1, 2]]
mi_var = l[0:2]    # mi_var vale [99, True]
mi_var = l[0:4:2]  # mi_var vale [99, "una lista"]
```

# Lenguaje python - Listas

Hay que mencionar así mismo que no es necesario indicar el principio y el final del slicing, sino que, si estos se omiten, se usarán por defecto las posiciones de inicio y fin de la lista, respectivamente:

```
l = [99, True, "una lista"]
mi_var = l[1:] # mi_var vale [True, "una lista"]
mi_var = l[:2] # mi_var vale [99, True]
mi_var = l[:] # mi_var vale [99, True, "una lista"]
mi_var = l[::2] # mi_var vale [99, "una lista"]
```

# Lenguaje python - Listas

También podemos utilizar este mecanismo para modificar la lista:

```
l = [99, True, "una lista", [1, 2]]  
l[0:2] = [0, 1] # l vale [0, 1, "una lista", [1, 2]]
```

pudiendo incluso modificar el tamaño de la lista si la lista de la parte derecha de la asignación tiene un tamaño menor o mayor que el de la selección de la parte izquierda de la asignación:

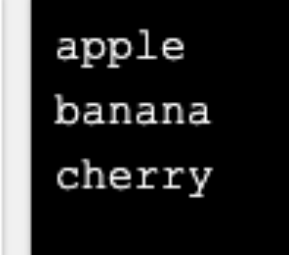
```
l[0:2] = [False] # l vale [False, "una lista", [1, 2]]
```

# Lenguaje python - Listas

## Recorrer una lista

Puede recorrer los elementos de la lista mediante un `for` bucle:

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```



```
apple  
banana  
cherry
```

# Lenguaje python - Listas

## Compruebe si el artículo existe

Para determinar si un elemento específico está presente en una lista, use la `in` palabra clave:

```
thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
    print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```



# Lenguaje python - Listas

## Longitud de lista

Para determinar cuántos elementos tiene una lista, use la `len()` función:

```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist))
```

3

# Lenguaje python - Listas

## Agregar elementos

Para agregar un elemento al final de la lista, use el método `append ()` :

```
thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

# Lenguaje python - Listas

## Insertar Elementos:

Para agregar un elemento en el índice especificado, use el método `insert ()` :

```
thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

# Lenguaje python - Listas

## Remover el artículo

Existen varios métodos para eliminar elementos de una lista:

```
thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)
```

```
['apple', 'cherry']
```

# Lenguaje python - Listas

## Ejemplo

El método pop() elimina el índice especificado (o el último elemento si no se especifica el índice):

```
thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)
```

```
['apple', 'banana']
```

# Lenguaje python - Listas

## Ejemplo

La `del` palabra clave elimina el índice especificado:

```
thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)
```

```
['banana', 'cherry']
```

# Lenguaje python - Listas

## Ejemplo

La `del` palabra clave también puede eliminar la lista por completo:

```
thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist) #this will cause an error because you have
succsesfully deleted "thislist".
```

```
Traceback (most recent call last):
  File "demo_list_del2.py", line 3, in <module>
    print(thislist) #this will cause an error because you have
NameError: name 'thislist' is not defined
```

# Lenguaje python - Listas

## Ejemplo

El `clear()` método vacía la lista:

```
thislist = ["apple", "banana", "cherry"]  
del thislist  
print(thislist) #this will cause an error because you have  
succsesfully deleted "thislist".
```

```
Traceback (most recent call last):  
  File "demo_list_del2.py", line 3, in <module>  
    print(thislist) #this will cause an error because you have  
NameError: name 'thislist' is not defined
```



# Lenguaje python - Listas

## Copiar una lista

No se puede copiar una lista simplemente escribiendo `list2 = list1`, ya que: `list2` sólo será una *referencia* a `list1`, y los cambios realizados en `list1` Automáticamente, también sean por `list2`.

Hay formas de hacer una copia, una forma es utilizar el método List integrado `copy()`.

```
thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

# Lenguaje python - Listas

Otra forma de hacer una copia es utilizar el método integrado **list()**.

```
thislist = ["apple", "banana", "cherry"]  
mylist = list(thislist)  
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

# Lenguaje python - Listas

## Unir dos listas

Hay varias formas de unir o concatenar dos o más listas en Python.

Una de las formas más sencillas es utilizar el `+` operador.

```
list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

# Lenguaje python - Listas

Otra forma de unir dos listas es agregando todos los elementos de list2 a list1, uno por uno:

```
list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
    list1.append(x)  
  
print(list1)
```



```
['a', 'b', 'c', 1, 2, 3]
```

# Lenguaje python - Listas

O puede usar el `extend()` método, cuyo propósito es agregar elementos de una lista a otra lista:

```
list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)  
|
```



```
['a', 'b', 'c', 1, 2, 3]
```

# Lenguaje python - Listas

El constructor list ()

También es posible usar el constructor `list ()` para hacer una nueva lista.

```
thislist = list(("apple", "banana", "cherry"))  
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

# Lenguaje python - Listas

## Métodos de lista

Python tiene un conjunto de métodos integrados que puede usar en listas.

Method	Description
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list